UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/706,515	11/12/2003	Fei Luo	BEAS-1339US2	7689

23910 7590 07/12/2006

FLIESLER MEYER, LLP
FOUR EMBARCADERO CENTER
SUITE 400
SAN FRANCISCO, CA 94111

EXAMINER	
ZHEN, LI B	

ART UNIT	PAPER NUMBER
2194	

DATE MAILED: 07/12/2006

Please find below and/or attached an Office communication concerning this application or proceeding.

Office Action Summary	Application No.	Applicant(s)	
	10/706,515	LUO ET AL.	
	Examiner	Art Unit	
	Li B. Zhen	2194	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *10 May 2006*.

2a)☐ This action is **FINAL**. 2b)☒ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-10,13,16-18 and 21-25* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-10,13,16-18 and 21-25* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☒ The specification is objected to by the Examiner.

10)☒ The drawing(s) filed on *12 November 2003* is/are: a)☒ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)
2) ☒ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☒ Information Disclosure Statement(s) (PTO-1449 or PTO/SB/08) Paper No(s)/Mail Date *5/10/06*.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .
5) ☐ Notice of Informal Patent Application (PTO-152)
6) ☐ Other: _____.

DETAILED ACTION

1. Claims 1 – 10, 13, 16 – 18, and 21 – 25 are pending in the application.

Continued Examination Under 37 CFR 1.114

2. A request for continued examination under 37 CFR 1.114, including the fee set forth in 37 CFR 1.17(e), was filed in this application after final rejection. Since this application is eligible for continued examination under 37 CFR 1.114, and the fee set forth in 37 CFR 1.17(e) has been timely paid, the finality of the previous Office action has been withdrawn pursuant to 37 CFR 1.114. Applicant's submission filed on 05/10/2006 has been entered.

Response to Arguments

3. Applicant's arguments with respect to the claims have been considered but are moot in view of the new ground(s) of rejection.

Information Disclosure Statement

4. The information disclosure statement (IDS) submitted on 05/10/2006 is in compliance with the provisions of 37 CFR 1.97. Accordingly, the examiner considered the information disclosure statement.

Oath/Declaration

5. The oath of declaration [submitted 03/29/2004] identifies the fourth inventor as "Rahul Rivastava", but the application data sheet [submitted 11/12/2003] identifies the fourth inventor as "Rahul Srivastava". The last name of the fourth inventor listed in the Oath of Declaration is inconsistent with the last name of the fourth inventor listed in the application data sheet. Examiner notes that the application data sheet on the record is submitted before the oath of declaration and it is unclear as to which last name is the correct one. Appropriated correction is required. The Office encourages the filing of an application data sheet or a supplemental application data sheet to correct a typographical or transliteration error in the spelling of an inventor's name, see MPEP 601.05.

Specification

6. The title of the invention is not descriptive. A new title is required that is clearly indicative of the invention to which the claims are directed.

7. The specification is objected to as failing to provide proper antecedent basis for the claimed subject matter. See 37 CFR 1.75(d)(1) and MPEP § 608.01(o). Correction of the following is required: the claims recite a machine-readable medium and the specification does not provide proper antecedent basis for the claimed subject matter. Although the specification discloses computer readable medium and storage medium, the specification does not disclose machine-readable medium. Therefore, the specification fails to provide proper antecedent basis for the claimed subject matter.

Claim Objections

7. Claims 3 and 9 are objected to because of the following informalities: abbreviations (JDBC, JMS and J2EE) should be defined. Appropriate correction is required.

Claim Rejections - 35 USC § 101

8. 35 U.S.C. 101 reads as follows:

> Whoever invents or discovers any new and useful process, machine, manufacture, or composition of matter, or any new and useful improvement thereof, may obtain a patent therefor, subject to the conditions and requirements of this title.

9. Claims 1-10, 13, 16-18 and 21-25 are rejected under 35 U.S.C. 101 because the claimed invention is directed to non-statutory subject matter.

Claims 1-10, 13, 16-18 and 21-25 recites a "machine readable medium" and the specification fails to provide antecedent bases for this limitation [see objection to the specification above]. Without antecedent basis for "machine readable medium", it is unclear if the limitation intended to be the same as the storage media described as part of the disclosed program product or whether it's intended to be broader than the disclosed storage media. It is believed that the limitation "machine readable medium" is intended to claim something broader than the disclosed storage media and cover signals, waves and other forms of transmission media, that carry instructions. Therefore, the limitation "machine readable medium" is not limited to physical articles or objects which constitute a manufacture within the meaning of 35 USC 101 and enable any functionality of the instructions carried thereby to act as a computer component and realize their functionality. As such, the claim is not limited to statutory subject matter and is therefore non-statutory.

Double Patenting

10. The nonstatutory double patenting rejection is based on a judicially created doctrine grounded in public policy (a policy reflected in the statute) so as to prevent the unjustified or improper timewise extension of the "right to exclude" granted by a patent and to prevent possible harassment by multiple assignees. A nonstatutory obviousness-type double patenting rejection is appropriate where the conflicting claims are not identical, but at least one examined application claim is not patentably distinct from the reference claim(s) because the examined application claim is either anticipated by, or would have been obvious over, the reference claim(s). See, e.g., *In re Berg*, 140 F.3d 1428, 46 USPQ2d 1226 (Fed. Cir. 1998); *In re Goodman*, 11 F.3d 1046, 29 USPQ2d 2010 (Fed. Cir. 1993); *In re Longi*, 759 F.2d 887, 225 USPQ 645 (Fed. Cir. 1985); *In re Van Ornum*, 686 F.2d 937, 214 USPQ 761 (CCPA 1982); *In re Vogel*, 422 F.2d 438, 164 USPQ 619 (CCPA 1970); and *In re Thorington*, 418 F.2d 528, 163 USPQ 644 (CCPA 1969).

A timely filed terminal disclaimer in compliance with 37 CFR 1.321(c) or 1.321(d) may be used to overcome an actual or provisional rejection based on a nonstatutory double patenting ground provided the conflicting application or patent either is shown to be commonly owned with this application, or claims an invention made as a result of activities undertaken within the scope of a joint research agreement.

Effective January 1, 1994, a registered attorney or agent of record may sign a terminal disclaimer. A terminal disclaimer signed by the assignee must fully comply with 37 CFR 3.73(b).

11. Claims 1-10, 13, 16, 21, 22 and 24 are provisionally rejected on the ground of nonstatutory obviousness-type double patenting as being unpatentable over claims 1-16 of copending Application No. 10/706216 [hereinafter APP216]. Although the conflicting claims are not identical, they are not patentably distinct from each other because APP216 is directed to the same invention for dynamically generating a wrapper object.

As to claims 1 and 21, APP216 teaches dynamically generating a wrapper object comprising: receiving a vendor class and superclass [claim 1, line 2 of APP216], performing reflection on the class [claim 1, line 3 of APP216], generating a wrapper class as a subclass of the superclass [claim 1, line 4 of APP216], wherein the wrapper

class comprises at least one of the vendor specific extension methods from the vendor class [claim 8 of APP216], generating a wrapper object as an instance of the wrapper class by instantiating the wrapper class [claim 1, lines 5-6 of APP216]; and providing the wrapper object to an application program [claim 1, line 7 of APP216], thereby providing the application program with access to specific extension methods [claim 8 of APP216]. As to obtaining specific extension methods defined within the class the reflection including retrieving meta information for allowing a server to generate a wrapper class that matches the vendor class, Examiner notes that Reflection is an operational feature of Java that allows meta information to be retrieved from code (p. 10, paragraph 0024 of applicant's specification). Since APP216 also recites Reflection [claim 1, line 3], it would have been obvious to a person of ordinary skill in the art at the time the invention was made that APP216's reflection function would also retrieve meta information. It would also have been obvious to a person of ordinary skill in the art at the time the invention was made to store instructions for executing the methods of APP216 on a machine-readable medium so the machine-readable medium carrying the instructions would cause the processor to perform the methods of APP216.

As to claims 2-9, these claims correspond to claims 2-9 of APP216.

As to claims 10, 22 and 24, these claims correspond to the combination of claims 2, 10-12, 14 and 15 of APP216. It would also have been obvious to a person of ordinary skill in the art at the time the invention was made to store instructions for executing the methods of APP216 on a machine-readable medium so the machine-

readable medium carrying the instructions would cause the processor to perform the methods of APP216.

As to claims 13 and 16, these claims correspond to claims 13 and 16 of APP216.

12. This is a provisional obviousness-type double patenting rejection because the conflicting claims have not in fact been patented.

Claim Rejections - 35 USC § 103

13. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

14. This application currently names joint inventors. In considering patentability of the claims under 35 U.S.C. 103(a), the examiner presumes that the subject matter of the various claims was commonly owned at the time any inventions covered therein were made absent any evidence to the contrary. Applicant is advised of the obligation under 37 CFR 1.56 to point out the inventor and invention dates of each claim that was not commonly owned at the time a later invention was made in order for the examiner to consider the applicability of 35 U.S.C. 103(c) and potential 35 U.S.C. 102(e), (f) or (g) prior art under 35 U.S.C. 103(a).

15. Claims 1 – 10, 13, 16 – 18, and 21 – 25 rejected under 35 U.S.C. 103(a) as being unpatentable over U.S. Patent No. 6,993,774 to Glass in view of U.S. Patent Application Publication No. 2004/0143835 to Dattke et al. [hereinafter Dattke, cited in the previous office action].

16. As to claim 1, Glass teaches the invention substantially as claimed including a machine-readable medium carrying one or more sequences of instructions for dynamically generating a wrapper object [dynamic generation of remote proxies; col. 6, lines 40 – 55], which instructions, when executed by one or more processors, cause the one or more processors to carry out the steps of:

receiving a vendor defined class [reads the associated class 252 from a class repository, col. 18, lines 56 – 63; Examiner notes that the specification does not specifically define a vendor class, therefore a vendor class is interpreted as class that provides services to other applications. The subject class as disclosed in Glass exists on a server system and provides services to clients, see col. 8, lines 1 – 12. Therefore, the subject class as disclosed in Glass corresponds to the claimed vendor defined class.] and superclass [determine each of subject class 19 superclasses...storage area for the name, superclasses; col. 8, lines 29 – 41];

performing reflection on the vendor class [invokes reflection engine 36 to determine information regarding subject class 19; col. 8, lines 1 – 12] to obtain specific methods defined within the vendor class [information determined by the reflection process includes...list of methods; col. 8, lines 11 – 30] the reflection including retrieving

meta information [information determined by the reflection process includes the following: name; list of implemented interfaces; list of methods; and superclass information; col. 8, lines 11 – 30] for allowing a server to generate a wrapper class that matches the vendor class [byte code generator 42 is to directly generate the executable code corresponding to JClass information 38. JClass information 38 is the definition of the Java class of which remote proxy object 22 is an instance; col. 9, lines 10 – 28];

generating a wrapper class as a subclass of the superclass [remote proxy for the subject object will inherit all of the variables and methods of its ancestors; col. 7, lines 58 – 67], wherein the wrapper class comprises at least one of the vendor specific methods from the vendor class [Remote proxy object 22 has an interface and list of methods identical to subject object 18; col. 5, lines 52 – col. 6, line 6];

generating a wrapper object as an instance of the wrapper class by instantiating the wrapper class [class loader 46 takes the generated bytes of remote proxy class 23 stored in memory and loads them into a class structure which then can be instantiated to create remote proxy object 22; col. 10, lines 1 – 10], thereby associating a relationship between the wrapper object and a vendor object [generated interface is associated with subject class 19; col. 8, lines 40 – 48]; and

providing the wrapper object to an application program [loads remote proxy class 23 onto client system 14; col. 11, lines 4 – 13], thereby providing the application program with access to specific methods [Communications between client application 108 and server object 110 proceed by client application 108 communicating with remote proxy 154 through its interface IProxy 152; col. 13, lines 25 – 40].

Although Glass teaches the invention substantially, Glass does not specifically teach a wrapper class comprising at least one of vendor specific extension methods from the vendor class.

However, Dattke teaches an application extension runtime environment for vendor objects [an application extension runtime environment 100 for standard applications; pp. 2-3, paragraph 0031], generating a wrapper class [dynamic proxy] comprising at least one of vendor specific extension methods [extension object] from the vendor [standard application; p. 4, paragraph 0038] class [generate a dynamic proxy for the an extension object implemented by the application extension; pp. 2-3, paragraph 0031], generating a wrapper object as an instance of the wrapper class by instantiating the wrapper class [p. 3, paragraph 0035], thereby associating a relationship between the wrapper object and a vendor object [the extension factory generates a dynamic proxy for the extension object (step 410) and returns the dynamic proxy to the method of the standard application requesting the extension object (step 415). The method of the standard application calls a method of the extension object through the dynamic proxy; p. 3, paragraph 0035].

It would have been obvious to a person of ordinary skill in the art at the time of the invention was made to combine Glass with Dattke because Dattke's teachings allows dynamic proxy classes to provide extensions to standard applications [p. 1, paragraph 0005 of Dattke] and permits standard applications to be modified as part of a future release without requiring any specific modifications to support existing application extension [p. 2, paragraph 0023 of Dattke]. Dynamic proxy classes can be used to

create a type-safe proxy object for a list of interfaces without requiring pre-generation of the class prior to compilation [p. 1, paragraph 0005 of Dattke] and provides the advantage of allowing customers of the standard application to customize the features of the standard application by providing customer-specific extensions for the features implemented by the standard application [p. 1, paragraph 0002 of Dattke].

17. As to claim 10, Glass teaches the invention substantially as claimed including a machine-readable medium carrying one or more sequences of instructions for processing an invocation at a dynamically generated wrapper [dynamic generation of remote proxies; col. 6, lines 40 – 55], which instructions, when executed by one or more processors, cause the one or more processor to carry out the steps of:

receiving, from an application program [Local object 20 may request access to subject object 18; col. 5, line 52 – col. 6, line 7], an invocation call directed to a wrapped vendor object [In order to isolate the distributed processing communication requirements from local object 20, a remote proxy object 22 may be created on server system 12 and loaded onto client system 14; col. 5, line 52 – col. 6, line 7];

initiating pre-processing by calling a pre-invocation handler configured to execute server-side code [Type object 204 forwards the message to the appropriate EJB function object 206 for preliminary processing; col. 15, lines 38 – 56];

calling the wrapped vendor object [Local object 20 communicates with remote proxy object 22 which then communicates with subject object 18; col. 5, line 52 – col. 6, line 7];

receiving a result from the wrapped vendor object [Reference object 158 decodes the result and passes it to remote proxy 154; col. 13, lines 40 – 58];

initiating post-processing by calling a post-invocation handler configured to execute post processing server-side tasks [Set of streamers 180 handles the encoding and transmission of arguments and results according to the communication protocol used by the receiving object; col. 14, lines 13 – 31]; and

providing the result to the application [Remote proxy 154 then makes the result available to client application 108; col. 13, lines 40 – 58] thereby enabling the application program to access vendor specific methods of the wrapped vendor object [Communications between client application 108 and server object 110 proceed by client application 108 communicating with remote proxy 154 through its interface IProxy 152; col. 13, lines 25 – 40].

Although Glass teaches the invention substantially, Glass does not specifically teach enabling the application program to access vendor specific extension methods of the wrapped vendor object.

However, Dattke teaches an application extension runtime environment for vendor objects [an application extension runtime environment 100 for standard applications; pp. 2-3, paragraph 0031], generating a wrapper class [dynamic proxy] comprising at least one of vendor specific extension methods [extension object] from the vendor [standard application; p. 4, paragraph 0038] class [generate a dynamic proxy for the an extension object implemented by the application extension; pp. 2-3, paragraph 0031], generating a wrapper object as an instance of the wrapper class by

instantiating the wrapper class [p. 3, paragraph 0035], thereby associating a relationship between the wrapper object and a vendor object [the extension factory generates a dynamic proxy for the extension object (step 410) and returns the dynamic proxy to the method of the standard application requesting the extension object (step 415). The method of the standard application calls a method of the extension object through the dynamic proxy; p. 3, paragraph 0035].

It would have been obvious to a person of ordinary skill in the art at the time of the invention was made to combine Glass with Dattke because Dattke's teachings allows dynamic proxy classes to provide extensions to standard applications [p. 1, paragraph 0005 of Dattke] and permits standard applications to be modified as part of a future release without requiring any specific modifications to support existing application extension [p. 2, paragraph 0023 of Dattke]. Dynamic proxy classes can be used to create a type-safe proxy object for a list of interfaces without requiring pre-generation of the class prior to compilation [p. 1, paragraph 0005 of Dattke] and provides the advantage of allowing customers of the standard application to customize the features of the standard application by providing customer-specific extensions for the features implemented by the standard application [p. 1, paragraph 0002 of Dattke].

18. As to claim 21, Glass teaches the invention substantially as claimed including a machine-readable medium carrying one or more sequences of instructions for enabling an application program to interface with a vendor application [Communications between client application 108 and server object 110 proceed by client application 108

communicating with remote proxy 154 through its interface IProxy 152; col. 13, lines 25 – 40], which instructions, when executed by one or more processors, cause the one or more processors to carry out the steps of:

receiving a vendor provided class used to interface with third party software [reads the associated class 252 from a class repository, col. 18, lines 56 – 63; Examiner notes that the specification does not specifically define a vendor class, therefore a vendor class is interpreted as class that provides services to other applications. The subject class as disclosed in Glass exists on a server system and provides services to clients, see col. 8, lines 1 – 12. Therefore, the subject class as disclosed in Glass corresponds to the claimed vendor defined class.];

preparing a wrapper object [invokes reflection engine 36 to determine information regarding subject class 19; col. 8, lines 1 – 12] for interfacing with vendor specific methods of the vendor provided class by reflecting the vendor provided class [information determined by the reflection process includes the following: name; list of implemented interfaces; list of methods; and superclass information; col. 8, lines 11 – 30] and a superclass [determine each of subject class 19 superclasses...storage area for the name, superclasses; col. 8, lines 29 – 41] to form a wrapper class [byte code generator 42 is to directly generate the executable code corresponding to JClass information 38. JClass information 38 is the definition of the Java class of which remote proxy object 22 is an instance; col. 9, lines 10 – 28] from which the wrapper object is instantiated [class loader 46 takes the generated bytes of remote proxy class 23 stored

in memory and loads them into a class structure which then can be instantiated to create remote proxy object 22; col. 10, lines 1 – 10]; and

providing the wrapper object to the application program [loads remote proxy class 23 onto client system 14; col. 11, lines 4 – 13], thereby enabling the application program capability to access vendor specific methods of the vendor application using the wrapper object [Communications between client application 108 and server object 110 proceed by client application 108 communicating with remote proxy 154 through its interface IProxy 152; col. 13, lines 25 – 40].

Although Glass teaches the invention substantially, Glass does not specifically teach a wrapper class comprising at least one of vendor specific extension methods from the vendor class.

However, Dattke teaches an application extension runtime environment for vendor objects [an application extension runtime environment 100 for standard applications; pp. 2-3, paragraph 0031], generating a wrapper class [dynamic proxy] comprising at least one of vendor specific extension methods [extension object] from the vendor [standard application; p. 4, paragraph 0038] class [generate a dynamic proxy for the an extension object implemented by the application extension; pp. 2-3, paragraph 0031], generating a wrapper object as an instance of the wrapper class by instantiating the wrapper class [p. 3, paragraph 0035], thereby associating a relationship between the wrapper object and a vendor object [the extension factory generates a dynamic proxy for the extension object (step 410) and returns the dynamic proxy to the method of the standard application requesting the extension object (step 415). The

method of the standard application calls a method of the extension object through the dynamic proxy; p. 3, paragraph 0035].

It would have been obvious to a person of ordinary skill in the art at the time of the invention was made to combine Glass with Dattke because Dattke's teachings allows dynamic proxy classes to provide extensions to standard applications [p. 1, paragraph 0005 of Dattke] and permits standard applications to be modified as part of a future release without requiring any specific modifications to support existing application extension [p. 2, paragraph 0023 of Dattke]. Dynamic proxy classes can be used to create a type-safe proxy object for a list of interfaces without requiring pre-generation of the class prior to compilation [p. 1, paragraph 0005 of Dattke] and provides the advantage of allowing customers of the standard application to customize the features of the standard application by providing customer-specific extensions for the features implemented by the standard application [p. 1, paragraph 0002 of Dattke].

19. As to claim 22, Glass teaches the invention substantially as claimed including a machine-readable medium carrying one or more sequences of instructions for processing an invocation at a dynamically generated wrapper [dynamic generation of remote proxies; col. 6, lines 40 – 55] enabling an application program to interface with a vendor application [col. 13, lines 25 – 40], which instructions, when executed by one or more processors, cause the one or more processors to carry out the steps of:

receiving, from an application program [Local object 20 may request access to subject object 18; col. 5, line 52 – col. 6, line 7], an invocation call directed to a wrapped

vendor object [In order to isolate the distributed processing communication requirements from local object 20, a remote proxy object 22 may be created on server system 12 and loaded onto client system 14; col. 5, line 52 – col. 6, line 7];

calling the wrapped vendor object [Local object 20 communicates with remote proxy object 22 which then communicates with subject object 18; col. 5, line 52 – col. 6, line 7];

receiving a result from the wrapped vendor object [Reference object 158 decodes the result and passes it to remote proxy 154; col. 13, lines 40 – 58]; and

providing the result to the application program [Remote proxy 154 then makes the result available to client application 108; col. 13, lines 40 – 58], thereby enabling the application program to access vendor specific methods of the wrapped vendor object [Communications between client application 108 and server object 110 proceed by client application 108 communicating with remote proxy 154 through its interface IProxy 152; col. 13, lines 25 – 40].

Although Glass teaches the invention substantially, Glass does not specifically teach enabling the application program to access vendor specific extension methods of the wrapped vendor object.

However, Dattke teaches an application extension runtime environment for vendor objects [an application extension runtime environment 100 for standard applications; pp. 2-3, paragraph 0031], generating a wrapper class [dynamic proxy] comprising at least one of vendor specific extension methods [extension object] from the vendor [standard application; p. 4, paragraph 0038] class [generate a dynamic proxy

for the an extension object implemented by the application extension; pp. 2-3, paragraph 0031], generating a wrapper object as an instance of the wrapper class by instantiating the wrapper class [p. 3, paragraph 0035], thereby associating a relationship between the wrapper object and a vendor object [the extension factory generates a dynamic proxy for the extension object (step 410) and returns the dynamic proxy to the method of the standard application requesting the extension object (step 415). The method of the standard application calls a method of the extension object through the dynamic proxy; p. 3, paragraph 0035].

It would have been obvious to a person of ordinary skill in the art at the time of the invention was made to combine Glass with Dattke because Dattke's teachings allows dynamic proxy classes to provide extensions to standard applications [p. 1, paragraph 0005 of Dattke] and permits standard applications to be modified as part of a future release without requiring any specific modifications to support existing application extension [p. 2, paragraph 0023 of Dattke]. Dynamic proxy classes can be used to create a type-safe proxy object for a list of interfaces without requiring pre-generation of the class prior to compilation [p. 1, paragraph 0005 of Dattke] and provides the advantage of allowing customers of the standard application to customize the features of the standard application by providing customer-specific extensions for the features implemented by the standard application [p. 1, paragraph 0002 of Dattke].

20. As to claim 24, Glass teaches the invention substantially as claimed including a machine-readable medium carrying one or more sequences of instructions for

processing an invocation at a dynamically generated wrapper [dynamic generation of remote proxies; col. 6, lines 40 – 55], which instructions, when executed by one or more processors, cause the one or more processors to carry out the steps of:

receiving, from an application program [Local object 20 may request access to subject object 18; col. 5, line 52 – col. 6, line 7], a method invocation call directed to a vendor object [In order to isolate the distributed processing communication requirements from local object 20, a remote proxy object 22 may be created on server system 12 and loaded onto client system 14; col. 5, line 52 – col. 6, line 7];

calling a wrapper object for processing the method invocation call [Local object 20 communicates with remote proxy object 22 which then communicates with subject object 18; col. 5, line 52 – col. 6, line 7] wherein the wrapper object has been dynamically generated [byte code generator 42 is to directly generate the executable code corresponding to JClass information 38. JClass information 38 is the definition of the Java class of which remote proxy object 22 is an instance; col. 9, lines 10 – 28] from a vendor class [Remote proxy object 22 has an interface and list of methods identical to subject object 18; col. 5, lines 52 – col. 6, line 6] to be associated with the vendor object [generated interface is associated with subject class 19; col. 8, lines 40 – 48];

initiating pre-processing by the wrapper object, wherein the wrapper object calls a pre-invocation handler configured to perform server side logic [Type object 204 forwards the message to the appropriate EJB function object 206 for preliminary processing; col. 15, lines 38 – 56];

forwarding the method invocation call to the vendor object by the wrapper object on behalf of the application program [Local object 20 communicates with remote proxy object 22 which then communicates with subject object 18; col. 5, line 52 – col. 6, line 7];

receiving a result of the method invocation call from the vendor object by the wrapper object [Reference object 158 decodes the result and passes it to remote proxy 154; col. 13, lines 40 – 58];

initiating post-processing by the wrapper object, wherein the wrapper object calls a post-invocation handler configured to perform server-side logic [Set of streamers 180 handles the encoding and transmission of arguments and results according to the communication protocol used by the receiving object; col. 14, lines 13 – 31]; and

providing the result to the application program [Remote proxy 154 then makes the result available to client application 108; col. 13, lines 40 – 58], thereby enabling the application program to access vendor specific methods of the vendor object [Communications between client application 108 and server object 110 proceed by client application 108 communicating with remote proxy 154 through its interface IProxy 152; col. 13, lines 25 – 40].

Although Glass teaches the invention substantially, Glass does not specifically teach enabling the application program to access vendor specific extension methods of the wrapped vendor object.

However, Dattke teaches an application extension runtime environment for vendor objects [an application extension runtime environment 100 for standard

applications; pp. 2-3, paragraph 0031], generating a wrapper class [dynamic proxy] comprising at least one of vendor specific extension methods [extension object] from the vendor [standard application; p. 4, paragraph 0038] class [generate a dynamic proxy for the an extension object implemented by the application extension; pp. 2-3, paragraph 0031], generating a wrapper object as an instance of the wrapper class by instantiating the wrapper class [p. 3, paragraph 0035], thereby associating a relationship between the wrapper object and a vendor object [the extension factory generates a dynamic proxy for the extension object (step 410) and returns the dynamic proxy to the method of the standard application requesting the extension object (step 415). The method of the standard application calls a method of the extension object through the dynamic proxy; p. 3, paragraph 0035].

It would have been obvious to a person of ordinary skill in the art at the time of the invention was made to combine Glass with Dattke because Dattke's teachings allows dynamic proxy classes to provide extensions to standard applications [p. 1, paragraph 0005 of Dattke] and permits standard applications to be modified as part of a future release without requiring any specific modifications to support existing application extension [p. 2, paragraph 0023 of Dattke]. Dynamic proxy classes can be used to create a type-safe proxy object for a list of interfaces without requiring pre-generation of the class prior to compilation [p. 1, paragraph 0005 of Dattke] and provides the advantage of allowing customers of the standard application to customize the features of the standard application by providing customer-specific extensions for the features implemented by the standard application [p. 1, paragraph 0002 of Dattke].

21. As to claim 2, Glass teaches the wrapper object is dynamically generated at runtime [col. 6, lines 40 – 55].

22. As to claim 3, Glass teaches the superclass is one of a pre-existing JDBC, JMS, or connector class [superclass remote proxies; col. 7, lines 56 – 67; examiner notes that the superclass remote proxy is a proxy that allows communications between a client and server object, therefore, the superclass remote proxies are also connector classes].

23. As to claim 4, Glass teaches the superclass includes logic to handle server side tasks [forwards the message to the appropriate EJB function object 206 for preliminary processing; col. 15, lines 38 – 56].

24. As to claim 5, Glass teaches the wrapper class is generated in bytecode [byte codes representing remote proxy class 23 are generated; col. 7, lines 20 – 35].

25. As to claim 6, Glass teaches bytecode is generated for vendor methods [byte codes representing remote proxy class 23 are generated; col. 7, lines 20 – 35] not implemented in the superclass [superclass remote proxies; col. 7, lines 56 – 67; examiner notes that the superclass remote proxies include all of the variables and methods of the subject class' ancestors, therefore, the subject class would include methods that are not implemented in the superclass].

26. As to claim 7, Glass teaches the bytecode is generated using hot code generation [byte code generator 42 takes general information regarding the needed Java object and directly generates executable Java code without the need for the intermediate step of creating a Java source file, col. 9, lines 27 – 50; generated class functionality placed in specialized function objects referred to as EJB function objects, col. 15, lines 1 – 17].

27. As to claim 8, Glass as modified teaches providing the wrapper object [dynamic generation of remote proxies; col. 6, lines 40 – 55] to an application program [loads remote proxy class 23 onto client system 14; col. 11, lines 4 – 13], enables the application program to access standard features [Communications between client application 108 and server object 110 proceed by client application 108 communicating with remote proxy 154 through its interface IProxy 152; col. 13, lines 25 – 40] defined by the superclass [superclass remote proxies; col. 7, lines 56 – 67] and non-standard vendor extensions defined by the vendor defined class [p. 3, paragraph 0035 of Dattke]. As to the motivation for combining the teachings of Glass and Dattke, see the rejection to claim 1 above.

28. As to claim 9, Glass teaches Enterprise Java Beans [col. 15, lines 1 – 16] and Enterprise JavaBeans technology is the server-side component architecture for Java Platform, Enterprise Edition (Java EE).

29. As to claim 13, Glass teaches the server-side code executed by the pre-invocation handler includes global transaction processing code [Preliminary common processing may include...transaction management; col. 15, lines 38 – 57].

30. As to claim 16, Glass teaches the post-processing server-side tasks include global transaction management [generated class functionality may include...transaction management; col. 15, lines 1 – 16].

31. As to claim 17, Glass teaches wherein providing the wrapper object to an application program enables the application to access wrapped vendor objects without requiring a relinking of the application and a vendor software package [byte code generator 42 generates the executable code representing JClass information 38 into remote proxy class 23. The method then proceeds to step 66 where class loader 46 loads remote proxy class 23 onto client system 14 where it is now available for use; col. 11, lines 3 – 12; examiner notes that the remote proxy class is loaded and available for use without requiring relinking of the application and a vendor software package].

32. As to claim 18, see the rejection to claim 17 above.

33. As to claim 23, Glass teaches dynamically generating the wrapper class in byte code that perfectly matches with the vendor class [JClass information 38 contains

subject object's 18 name, interfaces, methods, and the computer code necessary for communications within distributed object management system 16.... Byte code generator 42 reviews JClass information 38 and generates the corresponding byte codes; col. 9, lines 10 – 28].

34. As to claim 25, Glass teaches the server-side logic includes at least one of global transaction management [generated class functionality may include...transaction management; col. 15, lines 1 – 16], pooling, caching, tracing and profiling.

Conclusion

35. The prior art made of record and not relied upon is considered pertinent to applicant's disclosure.

U.S. Patent No. 6578191 to Boehme discloses a method for dynamic "event to method" adapter class generation.

U.S. Patent Application Publication No. 20020152210 to Johnson discloses a system for providing access to a plurality of disparate content repositories comprising a client application program interface.

U.S. Patent Application Publication No. 20030105883 to Gibbons discloses a method for allowing Java objects to communicate with .Net Remoting objects.

CONTACT INFORMATION

36. Any inquiry concerning this communication or earlier communications from the examiner should be directed to Li B. Zhen whose telephone number is (571) 272-3768. The examiner can normally be reached on Mon - Fri, 8:30am - 5pm.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, William Thomson can be reached on 571-272-3718. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

Li B. Zhen
Examiner
Art Unit 2194
July 6, 2006

LBZ